Situated Tangible Robot Programming

Yasaman S. Sefidgar, Prerna Agarwal, and Maya Cakmak
Computer Science & Engineering, University of Washington
185 Stevens Way, Seattle, WA 98195
einsian@gmail.com, prerna2@uw.edu, mcakmak@cs.washington.edu

ABSTRACT

This paper introduces *situated tangible robot programming*, whereby a robot is programmed by placing specially designed tangible "blocks" in its workspace. These blocks are used for annotating objects, locations, or regions, and specifying actions and their ordering. The robot compiles a program by detecting blocks and objects in its workspace and grouping them into instructions by solving constraints. We present a proof-of-concept implementation using blocks with unique visual markers in a pick-and-place task domain. Three user studies evaluate the intuitiveness and learnability of situated tangible programming and iterate the block design. We characterize common challenges and gather feedback on how to further improve the design of blocks. Our studies demonstrate that people can interpret, generalize, and create many different situated tangible programs with minimal instruction or with no instruction at all.

Keywords

End-User Robot Programming, Programming by Demonstration, Tangible Programming

1. INTRODUCTION

Programmability is a key advantage of industrial robots over traditional automation [16]. Robot arms, such as Universal Robot's UR5, KUKA's LWR, or ABB's YuMi, can be programmed to perform a wide variety of manipulation tasks. Indeed, these robots are currently deployed in various settings, performing tasks such as loading, unloading, sorting, kitting, packaging, assembly, or disassembly of a diverse set of items, from electronics to food. These robots are often programmed to perform their unique task in their specialized environment by highly trained experts, largely because the languages and interfaces for programming these robots are notoriously complex. For example, they usually require users to specify coordinate frames, and as UR Manaul puts it, "A problem with such frames is that a certain level of mathematical knowledge is required to be able to define such coordinate systems and also that it takes a considerable amount of time to do this, even for a person skilled in the art of robot programming and installation" [1].

HRI '17, March 06 - 09, 2017, Vienna, Austria

ISBN 978-1-4503-4336-7/17/03…$15.00

DOI: http://dx.doi.org/10.1145/2909824.3020240

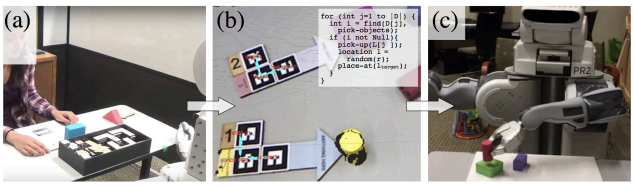

Figure 1: ***(a) Situated tangible programming*** **involves programming a robot by combining and placing specially designed tangible blocks in the robot's workspace to select objects, locations, or regions, and to specify actions and their ordering. (b) Blocks and objects in the workspace are detected by the robot and compiled into a robot program. (c) The robot can perform the instructed task in new environments and in the absence of blocks by executing this compiled program.**

The development of lower-cost and human-safe manipulators has increased the frequency of use of robotic arms by small and medium businesses in their manufacturing processes and even outside industrial contexts. As a result, robotics companies now place more emphasis on letting end-users program robots by themselves to reduce the cost of programming and eliminate down-time. For instance, Re-Think's robots, Baxter and Sawyer, are advertised as being "simple to train, flexible, and re-deployable"; Franka Emika's upcoming robot, dubbed "everybody's robot," offers "visually intuitive programming."

Despite these efforts by robotics companies and decades of research on robot Programming by Demonstration, many challenges remain in enabling non-experts to program robots. Referencing objects or arbitrary parts of the environment is particularly challenging for end-users. Most industrial tools support using the robot's arm to pre-record a set of coordinate frames. As noted, this separate step complicates the programming process and requires additional skills. While the proposed "situated" techniques, such as using pointing gestures [18, 8], possibly combined with verbal descriptions [14, 19, 7, 15, 27], simplify referencing objects and locations, they are not robust enough for industrial settings.

This paper proposes a new way to program robots that takes advantage of being situated in the task context. This approach involves placing physical, tangible "blocks" in the

robot's task environment to annotate objects, locations, or regions and to instruct the robot to perform actions that reference them. We first develop a programming language that supports programming of typical industrial robotic tasks and implement a proof-of-concept system with blocks that are easy to detect. We present findings from a user study (N=20) that examines the intuitiveness and learnability of situated tangible programming, and two follow-up studies, with an improved block design, that examine program interpretation (N=18) and creation (N=6) without *any* instructions. Our studies demonstrate that situated tangible programming offers a promising approach that lets novices program robots with minimal training and inform further improvements to the block design.

2. RELATED WORK

Our work contributes a new way of programming robots. Traditionally, industrial robots are programmed by saving a sequence of robot poses, assuming the position of all relevant items are known at programming time (e.g., a trash can that has a fixed position or parts that get placed at the same location and orientation by a feeder). Programming by Demonstration (PbD) extends this approach to programming capabilities that generalize to new situations. PbD has been widely researched, with work focusing on types of demonstrations [2], modalities for demonstration [29, 9], and alternative task representations [23, 21]. PbD work that focuses on manipulation tasks often involves a single object of interest in the workspace, avoiding the challenge of referencing objects. Exceptions include [3] and [24] that support limited referencing by automatically detecting objects involved in the task from a kinesthetic demonstration.

While PbD is the most popular technique in the literature, other ways in which end-users can program robots have been proposed. These include programming through natural language instructions (*e.g.,* [20, 7]) or visual programming (*e.g.,* [4, 13, 11]). Although the concept of programming manipulators with tangibles is novel, tangibles have been used for *controlling* robots, where the robot performs a task as long as the tangibles remain in its workspace. For example, Zhao *et al.* proposed placing paper tags ("Magic Cards") on the floor to task Roomba-like robots[30]. Or, Luria *et al.* introduced tangibles ("phicons") for communicating with a social robot to control smart home devices [17].

The promise of natural and direct interactions has made tangible interfaces appealing across varied application domains from music [22] and creative exploration [10] to math [26] and 3D modeling [5]. One controlled experiment with children provide evidence for benefits of tangible programming over visual programming [25]. Two different tangible programming systems developed for K-12 outreach were actually used for creating simple navigation programs for toy robots [12, 28]. Considering the situatedness inherent to many robot programming tasks, tangible programming is a promising candidate for accessible end-user programming of robot manipulators.

3. SITUATED TANGIBLE PROGRAMMING

We propose programming robots by placing tangible blocks in robot's physical workspace. To that end, we first develop the underlying programming language and design the blocks that allow creating programs in this language. Then, we present a proof-of-concept system implementation.

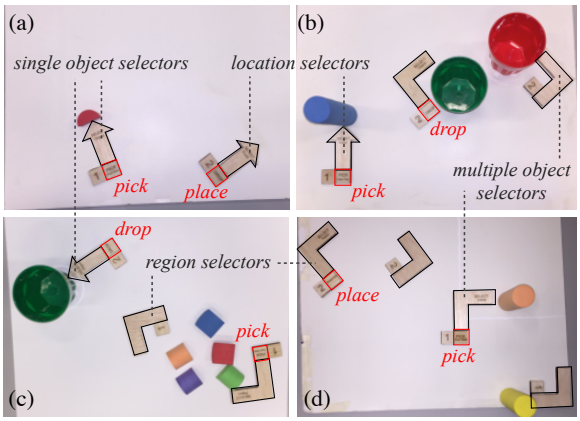

Figure 2: Example situated tangible programs in different scenes. Programs vary in the type of selection blocks used as part of the pick and place/drop instructions.

3.1 Domain and Programming Language

We focus on programming pick-and-place tasks in industrial settings. We assume the robot can detect the surfaces, where locations and regions of interest are defined, and the objects of interest on these surfaces (*e.g.,* parts and containers). The robot's actions are represented with four functions that can be called within a program: `pick-up-from-top`, `pick-up-from-side`, `place-at`, and `drop`. Actions take one of four types of arguments: a *location*, an *object*, a *region*, or a *list of objects*. A *location* is a 3-dimensional point on the robot's workspace and a *region* is a convex hull. Objects are represented with a *descriptor* that allows them to be localized in the robot's workspace.

In our framework, a *program* is a sequence of instructions that accomplish a certain pick-and-place task when executed by the robot. Programs can have *constants* whose values are determined at programming time. In addition each program has one *variable*, a list of all available objects, instantiated at execution time. Our language also has a number of convenience functions and control flow operators that allow looping or branching in the program.

3.2 Situated Tangible Blocks

Situated tangible programming involves creating programs in the language described in Sec. 3.1 using a set of physical *blocks*. Our design involves three categories of blocks.

Selection blocks ("selectors"). The first set of blocks allow selecting a single object, multiple objects, a location, or a region at programming time to instantiate a constant in the program. Fig. 2 shows examples of these blocks used as part of situated tangible programs. A single object is selected by an arrow placed on the surface carrying the object pointing towards it. A group of objects is selected by enclosing them between two 90^o corner brackets. A location on a surface is indicated by the tip of an arrow placed on the surface. A region on a surface is indicated by two corner brackets. Note that a region specified in this way will have four corners, but is not guaranteed to be a rectangle.

Action blocks. Action blocks correspond to the robot action functions described in Sec. 3.1. To instantiate the parameter of the function, the block is attached to a selection block. Fig. 2 shows how different action blocks are attached to selection blocks as part of a program. If a `pick` or `place/drop` action block is attached to a location selector (Fig. 2(a,b)), the action is performed at that location. A pick action requires that an object be found at that location at execution time, whereas the place/drop action will adjust the height of the action depending on whether the location is empty or already has an object.

If a `pick` or `place/drop` action block is attached to a single object selector (Fig. 2(a,c)), then the descriptor of that object is added as a constant input to the program. The program will locate this object in the new scene and perform the action at its location.

If a `pick` action block is attached to a region selector (Fig. 2(c)), the robot will pick up all objects found in that region at execution time. If a `place` action block is attached to a region selector (Fig. 2(d)), the robot will place the currently held object at a random location in that region.

If a `pick` action block is attached to a multiple object selector (Fig. 2(d)), then the resulting instruction will make the robot pick all instances of objects that are the same as an object in the selected set. This creates a loop in the program. If a `place/drop` action block is attached to a multiple object selector (Fig. 2(b)), then the resulting program will find an object in the current scene that matches any of the selected objects, and will place/drop the currently held object on it.

Ordering blocks. Ordering blocks have positive integers that indicate the order of instructions in the program. To compile a complete program from a set of blocks, our system uses the ordering constraints specified by these blocks, together with the constraint that each pick action needs to be paired with a place/drop action. Fig. 2 shows how ordering blocks are attached to action blocks.

From blocks to programs. Given a scene with a set of objects and blocks, the robot needs to first group blocks and objects together (see Sec. 3.4 for implementation details). A set of blocks form a valid program if: (a) each action block has a selection block and an ordering block associated with it, (b) all pick actions have odd number orderings and all place/drop actions have even number orderings (*i.e.,* the program starts with a pick and what is picked is placed/dropped before the next pick), and (c) object selectors have objects associated with them.

Loops are created whenever a pick action is associated with a *multiple object* or *region* selector. The place/drop action that follows the pick action is considered part of the loop. Conditionals are automatically added to the program for checking membership in lists of objects, checking existence, or checking location-region relationships.

3.3 Visual Block Design

The design of the language and blocks described above are based on the requirements of the functionality that we aim to provide. While it is important for blocks to be robustly perceivable by the robot, there is flexibility in how they look. Our design goal is to maximize intuitiveness and learnability of the blocks and combined instructions for end-users. To that end, we explore the following visual and physical design elements. Our studies involved two iterations of block design, which we refer to as D1 and D2.

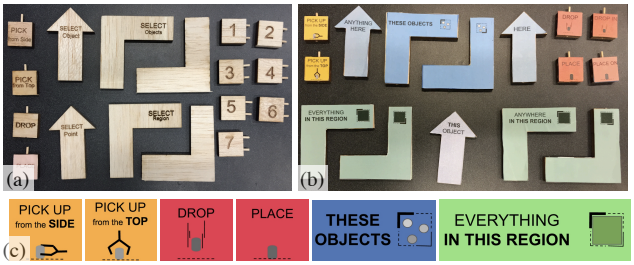

Figure 3: Situated tangible block libraries: (a) initial design (D1), (b) improved design (D2), (c) icons in D2.

Shape: Block shapes are chosen to reflect their functional category. Arrows are used for pointing to a location or to a single object to select it. Two L-shaped brackets are paired together to select an approximately rectangular region or a set of objects included in the region, analogous to using thumb and index fingers with two hands to indicate a region on a surface. Other blocks are square shaped, appropriately sized relative to the arrows and brackets, to allow combinations of blocks to look uniform.

Text and icons: Short word phrases and visual icons (D2 only) clarify the specific function of action or selection blocks. For instance, *single object* selectors and *location* selectors that have the same arrow shape, have different phrases on them. Action blocks have a verb phrase and ordering blocks have a single integer. The phrases used on D1 selectors are direct reflections of the formal terminology used here to describe different blocks. In contrast D2 uses natural language phrases. For instance, the single object selector has the phrase "select object" in D1 and "this object" in D2. Capitalization is used for emphasis as shown in Fig. 3. In D2, we also add *prepositions* to some action blocks ("place on," "drop in") and made separate location selectors for pick ("anything here") versus place ("here").

Color: Block colors (only available in D2) are also used to distinguish between block types that have the same shape. In addition, pairs of brackets were colored the same to help with association.

Peg/hole connectors: Blocks can be attached to one another through uniquely spaced peg-and-hole connectors that implicitly indicate the possibility of combining blocks while preventing false combinations.

3.4 System Implementation

We present a proof-of-concept implementation of situated tangible instructions.

Perception. Our system first detects all surfaces and objects of interest, as well as all tangible program blocks in the scene. Perception of blocks is simplified using fiducials (also known as AR Tags) incorporated into the block design. Our implementation uses the Alvar AR Tag tracking library.[1] Each type of block is given a unique identifier. Asymmetric AR Tags also allow uniquely determining the orientation of the blocks, allowing us to determine arrow and bracket orientations. Surfaces and objects are perceived using the table top segmentation functionality in the Point Cloud Library

[1] http://wiki.ros.org/ar_track_alvar

(PCL)[2]. This returns the dominant surface in the scene and a list of segmented objects on it. Objects are represented by their size, shape, and color.

Program compilation. Given a list of objects and lists of selection, action, and ordering blocks detected in the scene, our system produces a program with the following procedure. First, we iterate over all selection blocks to determine the program constants. For each *single object* selector, we iterate over the set of objects to determine which object is in the direction of the selector arrow on the surface within a certain distance. For each *multiple object* selector, we iterate over the set of objects to determine which objects are within the convex hull created by the selectors on the surface. Selected objects are saved in the program. Next, we iterate over all location and region selectors, and we save the constant location and region values as part of the program.

The next step is to iterate over all action blocks and assign them to a selector block. The peg/hole attachment mechanism on the blocks ensure that the displacement between a selection and action block follows a pattern and their orientations are aligned. These constraints allow uniquely identifying the assignment of action blocks to selection blocks. Next, we iterate over all ordering blocks and assign them to an action block. After grouping all objects and blocks, the final step is to compile the program as described in Sec. 3.2.

Actions and program execution. Our system is implemented on a PR2 robot (partially shown in Fig. 1), using the RGBD sensor mounted on the head and one of its 7 degree-of-freedom arms. The robot's actions are pre-defined as a sequence of poses relative to detected object locations. Pick actions have a pre-grasp, grasp, and lift pose determined based on the size of objects used in our evaluation and the type of pick (from top or from side). Place actions have a pre-placement, placement, and clear pose determined based on the size of objects. Drop actions have a single drop pose where the gripper is opened. All poses were pre-specified using kinesthetic demonstrations. During program execution, the robot determines the current set of objects and computes the poses for all actions that are relative to objects. The robot uses motion planning, as implemented in MoveIt![3] to move from one pose to the next.

4. EVALUATION

Following a human-centered approach, we evaluate the intuitiveness and learnability of situated tangible programming with users of varied backgrounds. We describe our evaluation spanning three studies; the first with our initial block design (D1) and the second and third with an improved design (D2) based on feedback from the first study.

4.1 Study 1: Initial Design

Programming languages and tools can be evaluated on three types of tasks: program comprehension, program creation, and debugging [6]. Our evaluation focus on the first two. To evaluate how well a program is comprehended, we employ two techniques: *demonstration* tests where we present participants with a program, situated in a scene, and ask them to demonstrate what the robot should do in that scene, and *generalization* tests in which we first show participants a program, situated in a *programming-time* scene, and next present them with robot's executions of that program in a different *execution-time* scene, and ask whether the execution is consistent with the program.

To evaluate how well people can use situated tangible blocks to create programs, we perform *program creation* tests where we describe the desired behavior of the program to be created. We provide participants with a set of objects and blocks and ask them to create a scene and a corresponding program with the described behavior.

4.1.1 Materials

Programs: Both demonstration and generalization tests involve programs with one pick and one place instruction where varied selector types are used for parametrizing the pick and place actions. Scenes and programs were prepared ahead of time and glued onto foam boards for consistency across participants. Scenes involved abstract objects varied in color, size, and shape (Fig. 2). In the demonstration test, participants are presented with different groups of four programs such that they see all selector and action types an equal number of times.

ID	Pick selector	Place/drop selector
P_1:	*location* selector	*single object* selector
P_2:	*single object* selector	*location* selector
P_3:	*location* selector	*region* selector
P_4:	*location* selector	*multiple object* selector
P_5:	*region* selector	*region* selector
P_6:	*multiple object* selector	*region* selector

Table 1: Programs used in the generalization tests. P_6 is also shown in Fig. 2(d).

Robot executions: In the generalization test, participants are presented with two executions of six different programs covering different combinations of selectors (Table 1). The executions occur in different scenes than those in which the program was presented to the participant. They differ in the objects involved in the scene and their placements. Seven out of 12 executions are fully consistent with the given program and five executions have an inconsistency (three during picking and two during placement). The executions are presented as a video of the robot manipulating objects in the scene (Fig. 4). Participants are asked to separately assess whether the pick actions and the place/drop actions in the execution are consistent with the given program, thus answering 24 yes/no questions presented randomly through a browser.

Program creation tasks: For program creation tests, participants are given a natural language description of three desired program behaviors and pictures illustrating the state of the scene before and after a sample execution of each (Fig. 5). P_A involves picking up an object placed at a particular corner of the table and placing it at the opposite corner. The intended program involves a pick and a place action, each parametrized with a *location* selector. P_B involves creating a stack of two specific objects (blue square at the bottom, red triangle on the top). The intended program involves two pick and place actions; the pick actions are parametrized with *single object* selectors and the place actions are parametrized with *location* selectors, though the placement of the second object on the first could also be

[2]http://pointclouds.org/
[3]http://moveit.ros.org/

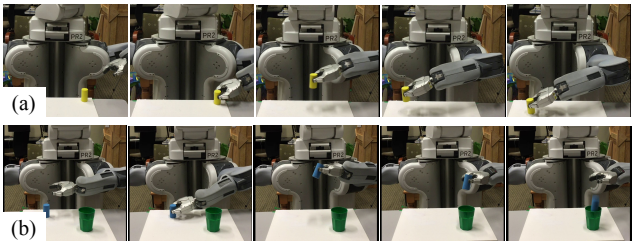

Figure 4: Filmstrips of example executions from the videos used in the generalization test. (a) A consistent execution of P_6. (b) An inconsistent execution of P_1.

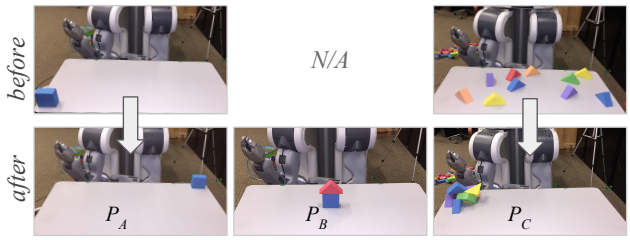

Figure 5: Before/after pictures used for describing the desired program behaviors in the program creation tests.

accomplished with a *single object* selector. P_C involves collecting triangles (of any color) and placing them near a particular corner of the table. The intended program involves a loop of pick and place actions, where the pick actions are parametrized with items from a *multiple object* selector and place actions are parametrized with a *region* selector.

4.1.2 Protocol

Our first study has three parts. In the first part, participants perform the demonstration and generalization tests without any instructions on what different blocks mean. In the second part, participants are first instructed about what each block means and how they can be combined. We then repeat the demonstration and generalization tests. In both parts we emphasize that objects are distinguished based on their shape, size, and color as the generalization test required such distinctions. Demonstration tests involve a different subset of programs, but the generalization questions are exactly the same. Order of programs in both tests are counter-balanced across participants. In the third part of the study, participants perform the program creation tasks.

4.1.3 Measurements

We record participants' demonstrations of each program in the demonstration tests, their yes/no answers and reasoning in the generalization tests, and their subjective ratings of confidence interpreting different tangible blocks in the form of 5-point Likert-style questions. In the second part, we additionally ask participants to compare their initial interpretations with our definitions and suggest alternatives that could have helped them better understand the intended meaning of each block. In the third part, we record programs created by participants and ask them to rate their experience programming with blocks.

Two authors double-coded videos of participants' demonstrations for consistency with the selection and action blocks as well as looping behavior in the program. Inconsistent demonstrations were further analyzed to identify common errors. Programs created by participants were scored for proper formation and placement of instructions and the overall program logic. Errors were noted and analyzed.

4.2 Follow-up Studies with Improved Design

Our follow up studies aim to assess the impact of the redesign on the intuitiveness of situated tangible programming without *any* instructions about the meanings of the blocks. To that end, Study 2 repeats Part 1 of Study 1 with the improved design (D2) to evaluate program comprehension. Study 3 evaluates program creation without any instruction and without having seen any example programs before. We give participants an abstract overview of situated tangible programming and then we ask them to brainstorm at least two interpretations for each block separately. Next, we ask them to connect blocks together and to brainstorm interpretations of the combinations. We do not give any feedback on their interpretations. Finally, we present them with the same program creation tests as in Study 1 in counterbalanced order.

5. FINDINGS

Participants for our studies were recruited from campus-wide mailing lists at the authors' university. Study 1 had 20 participants (12 F, 8 M); Study 2 had 18 participants (7 F, 11 M), and Study 3 had 6 participants (3 F, 3M). Study 1 took between 90 to 120 minutes, and Study 2 and 3 took 60 minutes.

We compare participants' performance on demonstration and generalization tests in two ways. First, we compare performance in Study 1, *before* and *after* being given instructions about the meanings of blocks. We refer to these two conditions as D1PRE and D1POST. Second, we compare performance before being given instructions for designs D1 (Study 1) versus D2 (Study 2). We refer to these as D1PRE and D2PRE. Note that the first comparison is within subjects and the second is between subjects. Hence, we perform paired t-tests between D1PRE and D1POST, and independent t-tests between D1PRE and D2PRE. The program creation tests in Study 1 and Study 3 are not comparable as they differ both in the design and the amount of instruction and practice provided prior to the task. We treat those conditions as observational studies.

5.1 Overall Performance

After minimal instructions, most participants were able to correctly interpret situated tangible programs for various tasks. In the D1POST condition, participants reached >90% across all metrics in the demonstration tests (Fig. 6(a)) and around 90% in the generalization tests (Fig. 6(b)). Almost all scores were significantly higher in D1POST compared to those achieved before being given instructions (D1PRE).

The redesign of the blocks (D2) significantly improved people's demonstrations in all aspects except in their interpretation of placement actions (already high with D1). Participants correctly interpreted around 80% of selection blocks attached to place/drop and above 90% of other components of programs in D2PRE. These are high scores to

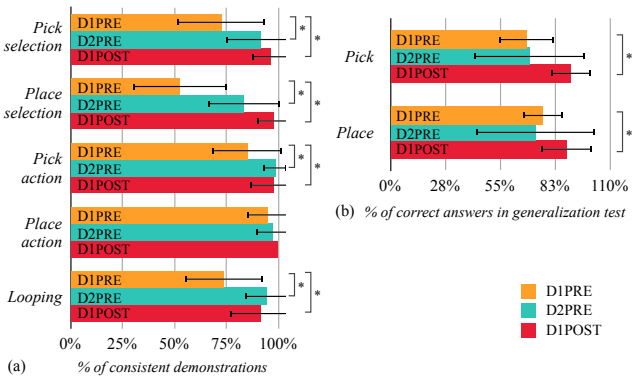

Figure 6: (a) Percentage of demonstrations (out of four) that were consistent with the program presented in the demonstration tests, in terms of interpretations of the selection and action blocks used for pick and place/drop as well as looping behavior, averaged across participants. (b) Percentage of correctly answered questions (out of 12) in the generalization test for pick and place portions of the robot's execution, averaged across participants. Statistically significant differences (p<0.05) are indicated with a *.

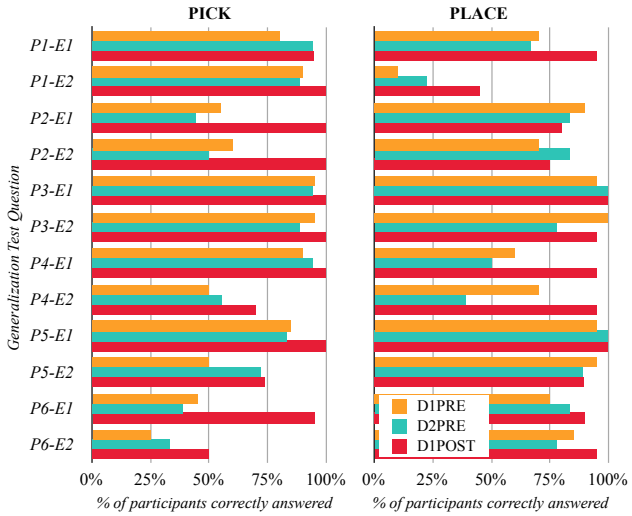

Figure 7: Percentage of participants who responded correctly to questions in the generalization test for two executions (E1, E2) of six programs (P_1-P_6); one question for pick and one question for place/drop. Three conditions are compared.

achieve without any instructions. However, people's ability to answer generalization questions did not improve with D2.

In Study 1, 70% of participants created correct programs for at least one program creation task. 40% successfully created correct programs for all three tasks. Program P_A (an abstraction of machine tending tasks) was the easiest (65% success rate), whereas P_B (an abstraction of assembly tasks) was the hardest (40% success rate). Achieving these rates with 5-minute instructions is considerable. In Study 3, in the absence of any instructions using D2, two out of six participants correctly programmed P_A and three participants correctly programmed P_B. One participant formed a valid program for P_A but the program did not satisfy the specified task (*i.e.,* it had an object selector instead of a location selector). None of the participants correctly programmed P_C; however, two had only a minor error.

5.2 Challenges with Program Comprehension

Table 2 summarizes the common challenges participants had in the demonstration tests. One of the most common challenges in D1PRE was in associating the two L-shaped brackets as parts of one selector. In D2, we tried to mitigate this issue by coloring paired brackets the same, varying the color of different bracket pairs, and including icons of a square region with opposite corners emphasized (Fig. 3). Another common issue was selecting only one object from a *multi-object* or *region* selector, rather than looping through all objects. In D2, we tried to mitigate this issue by changing the wording of the selectors to say "*everything* in this region" and "*these* objects". The other issues were due to alternative interpretations of words used on action and selection blocks (*e.g.,* "select" interpreted as an action verb), as well as their shape (*e.g.,* arrow as direction). In D2, we tried to mitigate these issues by updating the wordings; for instance, the term "select" was removed from selection blocks and replaced with referential phrases such as "this object" or "here." To further clarify the actions, we included icons of a gripper picking up an object from the side versus top.

Overall the generalization test was more challenging and it revealed additional confusions people had. To better understand challenges in the generalization test, we investigated the particular questions that participants tended to get wrong and examined the explanations of their answers. Fig. 7 shows participant's correct responses in the generalization test broken down into individual questions. The most common issue we observed was in differentiating object selectors from location/region selectors. We noted at least one instance of confusion in this regard (verified by participant's explanation) in 90%, 83.3%, and 45% of participants in D1PRE, D2PRE, and D1POST respectively.

For example, one question about P_2 (Table 1) involved the robot picking up the selected object from a different location. This is consistent with the program; however, 50% of participants in D1PRE deemed it inconsistent (Fig. 7(a), P_2-E1/E2). Further examination of their explanations revealed flawed mental models of object selectors. For example, one participant said *"The object's initial location does not seem to be at all near to the indicated point, so the robot should not grab this object."* Similarly, around 20% of participants in D1PRE thought that the robot's execution of P_1 picking a different object at the selected location was inconsistent with the program (Fig. 7(a), P_1-E1/E2); *e.g., "It violated the picking instruction because it picked up a green cylinder even though instruction #1 has an arrow with a command "pick from side" pointing to a purple cylinder."* Nonetheless, many participants gave correct answers backed by explanations that demonstrate correct mental models; *e.g., "The instruction specifies the object, which the robot finds and picks up"* (P_2, object selector) and *"Yes, the robot picks an item from the correct location"* (P_1, location selector).

Our examination of participants' responses identified additional mental model flaws that contributed to wrong answers. For example, P_1 included an object selector pointing

Error Description	D1PRE	D2PRE	D1POST
Failure to associate paired brackets as opposite corners	65%	11.1%	0%
Failure to loop over multiple objects when picking from a region or a set of objects	60%	0%	30%
Misinterpretation of "side" and "top" not in relation to objects	35%	0%	0%
Misinterpretation of the word "select" as an action	35%	NA	0%
Misinterpretation of arrow selectors as referring to a direction	10%	16.7%	0%
Misinterpretation of icons	NA	33.3%	NA
Placement on all objects for the multi-object selector with the words "these objects"	NA	16.7%	NA
Misinterpretation of "anything here" as a reference to $\geqslant$1 object	NA	11.1%	NA

Table 2: Different user errors in the demonstration tests and percentage of participants who made the error at least once. Note that numbers in the first and third column are out of 20 (Study 1) and the second column is out of 18 (Study 2) participants.

to a purple cup, with the drop action attached to it. In an inconsistent execution, the robot dropped an object in a green cup. Only 25% of the participants in D1PRE judged this execution as inconsistent with P_1; this went up to only 33% in D2PRE. Even after being explicitly instructed about how the robot represents objects and how the object selector matches all properties of objects in D1POST, around 50% of participants still thought the robot's action was consistent with the program. The explanations of their answers revealed that even though they had a correct mental model of object selectors, they viewed the two cups as the same objects, likely because of their common affordance in the context of dropping. One participant said *"The green cup is clearly a close analogue for the purple cup, so I think dropping into the green cup is perfectly reasonable."*

Fig. 8 shows participants' responses to the Likert-scale question asking how much their initial mental model of a block matched its actual meaning after getting instructions about blocks. We see that their perceived mental model discrepancy is consistent with challenges observed prior to instructions. Actions were interpreted correctly by most participants. People's pre-instruction mental model of location selectors were more likely to be correct than those of object selectors. Also, the region and multiple object selectors were more likely to be misinterpreted.

5.3 Challenges with Program Creation

Error Description	#
Failure to distinguish between object and location selectors	5
Misuse of object selectors without the selected objects	6
Missing ordering on brackets	5
Unattached ordering blocks in attempt to reuse selection and action blocks in an earlier instruction	2

Table 3: Different user errors in the program creation test (post instructions on D1) and the number of participants (out of 20) who made that error.

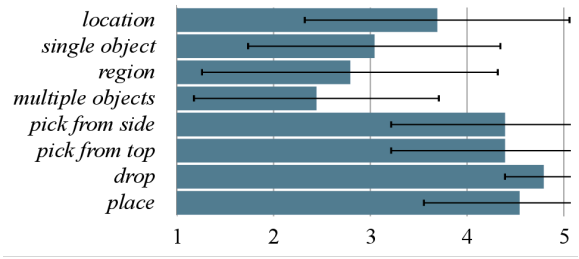

Figure 8: Responses to 5-point Likert-scale questions asking participants to rate how well their pre-instruction mental model of different block types matched their post-instruction mental model (1: very different).

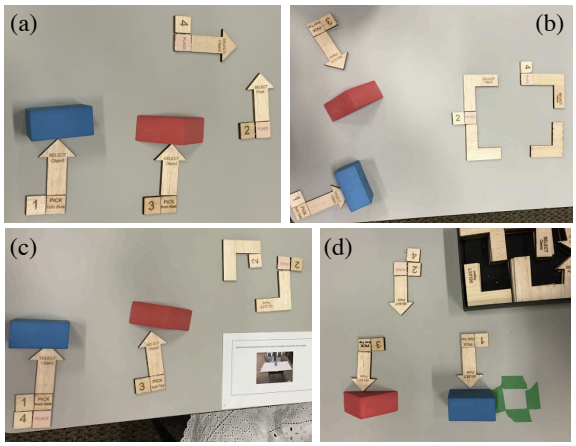

Figure 9: Alternative programs created by participants in the program creation task for P_B.

Table 3 summarizes challenges observed with program creation with D1. There were three errors that were observed in at least 5 of 20 participants. The first was improper use of object and location selectors, indicating the persistence of mental model flaws observed in demonstration and generalization tests. The second concerned improper placement of object selectors. Some of these were byproducts of the first type of error: participants who confused object and location selectors did not realize objects must be present with object selectors. Others assumed the robot could simulate the effect of pick and place actions and they used the object selector where the object should appear during the execution of the program (most common in task P_B). A program created by a participant with this error is shown in Fig. 9(a). The third common error was missing numbering on one of the brackets, which was required for distinguishing pairing of brackets when more than one pair was used in the program.

Table 4 summarizes the common errors observed in program creation test in Study 3 (using D2). We observe some of the same issues. Once again, participants did not make the distinction between object and location selectors. They commonly used "this object" interchangeably with "anything here" or "everything in this region" interchangeably with "these objects". Failing to number all brackets was also common. We additionally observed unexpected combinations of blocks. Two participants formed regions using addi-

Error Description	#
Failure to distinguish between object and location selectors	4
Missing ordering on brackets	4
Forming regions with >2 brackets (typically 4)	2
Combining arrows and bracket selectors	2

Table 4: Different errors in the program creation test using D2 and the number of participants (out of 6) who made that error.

tional brackets. Participants who incorrectly used additional brackets attributed their choice to the icons on the brackets. Two participants combined arrows with brackets to identify where the arrow is pointing. They thought that arrows indicate direction and that the bracket confines where the arrow points.

Fig. 9 shows different programs created by participants for the same task (P_B) using D1. The program in Fig. 9(a) is invalid as it places an object selector at a "future" location of the object. This error was avoided in Fig. 9(c), by attaching the place action to the object selector at the object's initial pose. Programs in Fig. 9(b) and (c) have alternative interpretations of the task description (*e.g.,* the assembly being at an exact location versus any point in a region), though these could result in the top part not being placed on the bottom part. This issue is avoided in Fig. 9(d) by using an exact location selector for placing both objects.

5.4 Impact of the Design Iteration

As mentioned in Sec. 5.1, participants performed significantly better in D2PRE compared to D1PRE in demonstration tests (Fig. 6). D2 appears to have successfully addressed issues with associating the paired brackets, misinterpretation of phrases, and looping (Table 2). However, misinterpretation of arrows as direction indicators persisted with D2. In addition, modified wordings and icons led to new confusions, albeit much less common. In their explanations during demonstrations, some participants noted matched coloring of paired brackets and the icon helped them understand that they represent a *box* together. Wordings on multi-object selectors attached to pick actions also proved helpful, as many participants commented they picked all objects in a loop because it says "everything" or "these". However, the same wording was problematic when attached to place actions. Some participants interpreted "these objects" as requiring placement on *all* objects. Despite doing better on demonstration tests, participants still had issues in generalization tests. Similar to D1, the dominant mental model flaw was treating all selectors as location selectors.

5.5 Impact of Instructions

Participants did significantly better on both demonstration and generalization tests knowing the meaning of blocks and how they form instructions. However, some issues remained even after instructions. Around 10% of participants still failed to loop over all objects when a pick action was attached to a region selector in the demonstration tests. In the generalization tests, mental model flaws associated with object versus location/region selectors were not fully addressed. In the program creation tests, instructions allowed avoiding unexpected combinations of blocks, such as using additional brackets or combining arrows and brackets.

6. DISCUSSION

Our studies demonstrate that situated tangible programming allows participants to program a robot with minimal or no instruction. We cannot draw a direct comparison with instructions given to participants for other end-user robot programming techniques in the literature; however, the ability for participants to program the robot without *any* instructions is unique.

The intuitiveness of situated tangible programming comes at the cost of expressiveness. Traditional text-based or visual programming languages allow much more complex boolean expression computations and flexible use of loops and conditionals. Programming by Demonstration allows defining new actions, whereas our approach is limited to preprogrammed actions. Nonetheless, our implementation captures many variations of pick and place tasks common in industrial settings.

The expressiveness of situated tangible programming could be increased by introducing additional blocks, such as customizable conditionals and loops. However, designing different blocks to be intuitive is challenging, as demonstrated in our studies. It is also possible to use situated tangible programming in conjunction with other end-user programming techniques. For instance, the user could define new actions with Programming by Demonstration, assign them to a block, and then use that block in a situated program.

Our studies involved programming without any system feedback or testing. Other end-user programming techniques often have *development environments* that aid the programming and debugging process. Similarly, we envision our system to provide feedback to users during programming. This could involve visualization of robot's perception, text representation of a program, and intuitive compile errors.

The most common challenge observed in our study was confusion between location and object selectors. This issue persisted across three studies with both designs in program comprehension and creation. This might have been because the shape of the block was much more salient than the color or text on it. Participants thought that the two selectors were the same because they were both arrows. This suggests semantically different blocks should have different shapes.

7. CONCLUSION

We introduce situated tangible programming: a new way of programming robots by placing tangible "blocks" in the robot's workspace. Blocks allow annotating objects, locations, or regions in the environment and specifying actions to be performed on them. We iteratively designed the blocks and introduced a proof-of-concept implementation of the system. Our evaluation shows that people can successfully comprehend and create many programs, even without any instructions. We demonstrate that design iterations can help improve intuitiveness of blocks and that people can interpret most programs correctly after minimal instructions. We identify common mental model flaws that lead to misinterpretations or creation of wrong programs, and we suggest possible improvements to address them.

Acknowledgements

This work was supported by the National Science Foundation Award IIS-1552427 "CAREER: End-User Programming of General-Purpose Robots."

8. REFERENCES

[1] Universal robots user manual. Universal robots website. https://universal-robots.com/media/8704/ur5_user_manual_gb.pdf, 2014.

[2] B. Akgun, M. Cakmak, J. W. Yoo, and A. L. Thomaz. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398, 2012.

[3] S. Alexamdrova, M. Cakmak, K. Hsaio, and L. Takayama. Robot programming by demonstration with interactive action visualizations. In *Robotics: Science and Systems (RSS)*, 2014.

[4] S. Alexandrova, Z. Tatlock, and M. Cakmak. Roboflow: A flow-based visual programming language for mobile manipulation tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5537–5544. IEEE, 2015.

[5] D. Anderson, J. L. Frankel, J. Marks, D. Leigh, E. Sullivan, J. Yedidia, and K. Ryall. Building virtual structures with physical blocks. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 71–72. ACM, 1999.

[6] M. Burnett. Visual programming. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 1999.

[7] R. Cantrell, P. Schermerhorn, and M. Scheutz. Learning actions from human-robot dialogues. In *2011 RO-MAN*, pages 125–130. IEEE, 2011.

[8] R. Fang, M. Doering, and J. Y. Chai. Embodied collaborative referring expression generation in situated human-robot interaction. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pages 271–278. ACM, 2015.

[9] K. Fischer, F. Kirstein, L. C. Jensen, N. Kr, K. Kukli, M. V. aus der Wieschen, T. R. Savarimuthu, et al. A comparison of types of robot control for programming by demonstration. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 213–220. IEEE, 2016.

[10] D. Gallardo, C. F. Julià, and S. Jorda. Turtan: A tangible programming language for creative exploration. In *Tabletop*, pages 89–92. Citeseer, 2008.

[11] D. F. Glas, T. Kanda, and H. Ishiguro. Human-robot interaction design using interaction composer: Eight years of lessons learned. In *The Eleventh ACM/IEEE International Conference on Human Robot Interation*, pages 303–310. IEEE Press, 2016.

[12] M. S. Horn and R. J. Jacob. Designing tangible programming languages for classroom use. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 159–162. ACM, 2007.

[13] J. Huang, T. Lau, and M. Cakmak. Design and evaluation of a rapid programming system for service robots. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 295–302. IEEE, 2016.

[14] T. Kollar, S. Tellex, D. Roy, and N. Roy. Toward understanding natural language directions. In *Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on*, pages 259–266. IEEE, 2010.

[15] S. Lauria, G. Bugmann, T. Kyriacou, and E. Klein. Mobile robot programming using natural language. *Robotics and Autonomous Systems*, 38(3):171–181, 2002.

[16] T. Lozano-Perez. Robot programming. *Proceedings of the IEEE*, 71(7):821–841, 1983.

[17] M. Luria, G. Hoffman, B. Megidish, O. Zuckerman, and S. Park. Designing vyo, a robotic smart home assistant: Bridging the gap between device and social agent. In *Robot and Human Interactive Communication (RO-MAN), 2016 25th IEEE International Symposium on*, pages 1019–1025. IEEE, 2016.

[18] C. Matuszek, L. Bo, L. Zettlemoyer, and D. Fox. Learning from unscripted deictic gesture and language for human-robot interactions. In *AAAI*, pages 2556–2563, 2014.

[19] C. Matuszek, D. Fox, and K. Koscher. Following directions using statistical machine translation. In *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*, pages 251–258. IEEE Press, 2010.

[20] S. Mohn and J. Laird. Learning goal-oriented hierarchical tasks from situated interactive instruction. In *Proceedings of the The Twenty-eighth National Conference on Artificial Intelligence (AAAI)*, 2014.

[21] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller. Interactive hierarchical task learning from a single demonstration. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pages 205–212. ACM, 2015.

[22] H. Newton-Dunn, H. Nakano, and J. Gibson. Block jam: a tangible interface for interactive music. In *Proceedings of the 2003 conference on New interfaces for musical expression*, pages 170–177. National University of Singapore, 2003.

[23] S. Niekum, S. Chitta, A. Barto, B. Marthi, and S. Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9, 2013.

[24] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5239–5246, 2012.

[25] T. Sapounidis, S. Demetriadis, and I. Stamelos. Evaluating children performance with graphical and tangible robot programming tools. *Personal and Ubiquitous Computing*, 19(1):225–237, 2015.

[26] E. Schweikardt and M. D. Gross. roblocks: a robotic construction kit for mathematics and science education. In *Proceedings of the 8th international conference on Multimodal interfaces*, pages 72–75. ACM, 2006.

[27] L. She, Y. Cheng, J. Y. Chai, Y. Jia, S. Yang, and N. Xi. Teaching robots new actions through natural language instructions. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 868–873. IEEE, 2014.

[28] A. Sipitakiat and N. Nusen. Robo-blocks: designing debugging abilities in a tangible programming system for early primary school children. In *Proceedings of the 11th International Conference on Interaction Design and Children*, pages 98–105. ACM, 2012.

[29] M. Tykal, A. Montebelli, and V. Kyrki. Incrementally assisted kinesthetic teaching for programming by demonstration. In *The Eleventh ACM/IEEE International Conference on Human Robot Interation*, pages 205–212. IEEE Press, 2016.

[30] S. Zhao, K. Nakamura, K. Ishii, and T. Igarashi. Magic cards: a paper tag interface for implicit robot control. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 173–182. ACM, 2009.